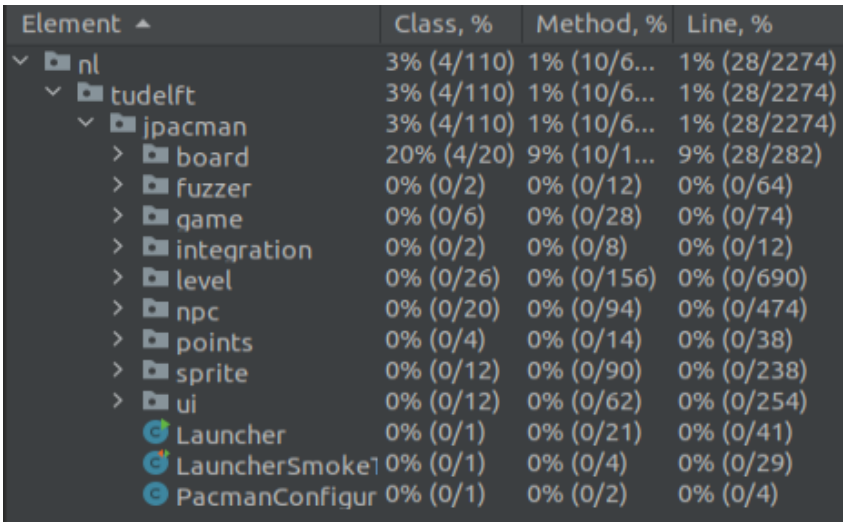
Testing Lab Report

By: Zach Conlin

Task 1 – Initial Test Coverage

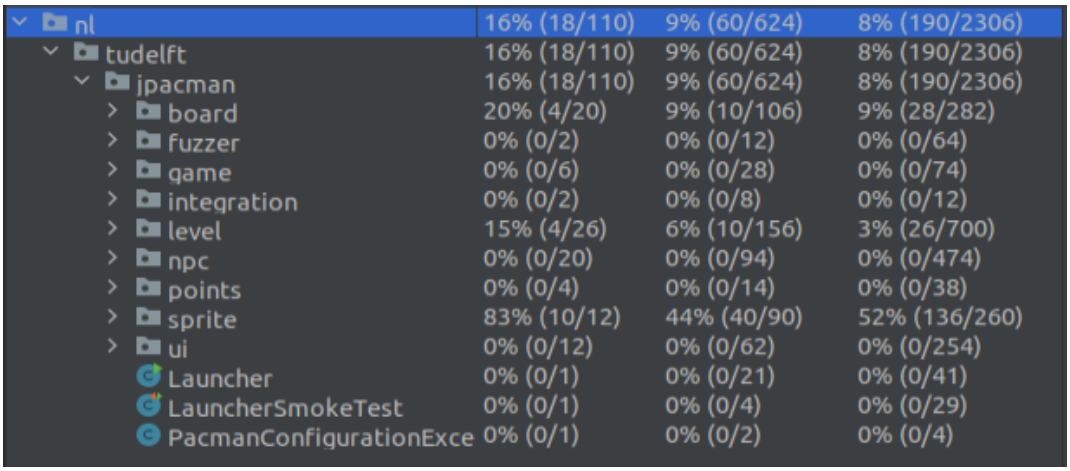
Initially, the test coverage is minimal; it is not enough to make any determination as to whether or not the code is working as expected.



Element	Class, %	Method, %	Line, %
nl	3% (4/110)	1% (10/6...	1% (28/2274)
tudelft	3% (4/110)	1% (10/6...	1% (28/2274)
jpacman	3% (4/110)	1% (10/6...	1% (28/2274)
board	20% (4/20)	9% (10/1...	9% (28/282)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
game	0% (0/6)	0% (0/28)	0% (0/74)
integration	0% (0/2)	0% (0/8)	0% (0/12)
level	0% (0/26)	0% (0/156)	0% (0/690)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	0% (0/12)	0% (0/90)	0% (0/238)
ui	0% (0/12)	0% (0/62)	0% (0/254)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmoke1	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigur	0% (0/1)	0% (0/2)	0% (0/4)

Task 2 – Increasing the Coverage

Once the test for the function isAlive() was implemented, coverage greatly increased. As indicated by the coverage per package below, gains in the sprite and level code coverage mainly contributed to this rise in the overall coverage.



nl	16% (18/110)	9% (60/624)	8% (190/2306)
tudelft	16% (18/110)	9% (60/624)	8% (190/2306)
jpacman	16% (18/110)	9% (60/624)	8% (190/2306)
board	20% (4/20)	9% (10/106)	9% (28/282)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
game	0% (0/6)	0% (0/28)	0% (0/74)
integration	0% (0/2)	0% (0/8)	0% (0/12)
level	15% (4/26)	6% (10/156)	3% (26/700)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	83% (10/12)	44% (40/90)	52% (136/260)
ui	0% (0/12)	0% (0/62)	0% (0/254)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationExce	0% (0/1)	0% (0/2)	0% (0/4)

Test coverage was further increased by implementing tests for the methods makeGhostSquare() in MapParser.java, createGhost() in LevelFactory.java, and isAnyPlayerAlive() in Level.java.

The test for makeGhostSquare() simply checks to make sure that the map parser creates the proper ghost square on the map after initializing all objects required to make the map parser class.

```
public class MapParseTest {  no usages
    PacManSprites testSprites = new PacManSprites();  3 usages
    GhostFactory testGhostFactory = new GhostFactory(testSprites);  2 usages
    Ghost testGhost = testGhostFactory.createBlinky();  1 usage
    List<Ghost> testGhosts = new ArrayList<>();  1 usage
    DefaultPointCalculator testPointCalculator = new DefaultPointCalculator();  1 usage
    LevelFactory testLevelFactory = new LevelFactory(testSprites, testGhostFactory, testPointCalculator);  1 usage
    BoardFactory testBoardFactory = new BoardFactory(testSprites);  1 usage
    MapParser testParser = new MapParser(testLevelFactory,testBoardFactory);  1 usage
    @Test  no usages
    void testMakeGhostSquare(){
        assert testParser.makeGhostSquare(testGhosts,testGhost).getOccupants().get(0).hasSquare();
    }
}
```

The test for createGhost() makes sure that the function creates all of the ghosts in the order specified by the ghostIndex field in LevelFactory.

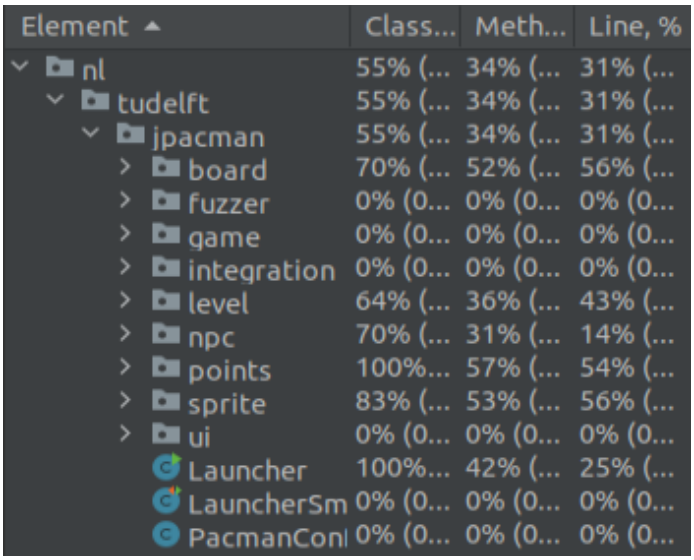
```
public class LevelFactoryTest {  no usages
    PacManSprites testSprites = new PacManSprites();  2 usages
    GhostFactory testGhostFactory = new GhostFactory(testSprites);  5 usages
    DefaultPointCalculator testPointCalculator = new DefaultPointCalculator();  1 usage
    Ghost testBlinky = testGhostFactory.createBlinky();  1 usage
    Ghost testInky = testGhostFactory.createInky();  1 usage
    Ghost testPinky = testGhostFactory.createPinky();  1 usage
    Ghost testClyde = testGhostFactory.createClyde();  1 usage
    LevelFactory testLevelFactory = new LevelFactory(testSprites,testGhostFactory,testPointCalculator);  4 usages
    @Test  no usages
    void testCreateGhost(){
        assert testLevelFactory.createGhost().getClass().equals(testBlinky.getClass());
        assert testLevelFactory.createGhost().getClass().equals(testInky.getClass());
        assert testLevelFactory.createGhost().getClass().equals(testPinky.getClass());
        assert testLevelFactory.createGhost().getClass().equals(testClyde.getClass());

    }
}
```

The test for isAnyPlayerAlive() first checks to see if initially there is any player alive; at this stage, there should not be. Then after registering a player in the game, the test checks again if there are any players currently alive. Once this occurs, the function should return true.

```java
public class LevelTest {  no usages
    PacManSprites testSprites = new PacManSprites();  1 usage
    PlayerFactory testPlayerFactory = new PlayerFactory(testSprites);  1 usage
    Player testPlayer = testPlayerFactory.createPacMan();  1 usage
    Launcher testLauncher = new Launcher();  1 usage
    Level testLevel = testLauncher.makeLevel();  3 usages
    @Test  no usages
    void testisAnyPlayerAlive(){
        assert !testLevel.isAnyPlayerAlive();
        testLevel.registerPlayer(testPlayer);
        assert testLevel.isAnyPlayerAlive();
    }
}
```

Below is the final test coverage after implementing the isAlive() test and the three above tests.



Task 3 – JaCoCo Test Coverage

In JaCoCo, it seems like the coverage is larger than the figures presented in IntelliJ. The reason for the difference is that IntelliJ checks the test code in addition to the main source code; Because there are no tests for the tests they all detract from the total test coverage.

For quick analysis of the test coverage, IntelliJ's visualization provides the figures in a convenient way. However, I found JaCoCo's report much more detailed in the case I wanted to see exactly where the test coverage numbers came from. The branch test coverage feature in JaCoCo is definitely helpful in testing conditionals in the code.

jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		67%		57%	74	155	104	344	21	69	4	12
nl.tudelft.jpacman.npc.ghost		71%		55%	56	105	43	181	5	34	0	8
nl.tudelft.jpacman.ui		77%		47%	54	86	21	144	7	31	0	6
default		0%		0%	12	12	21	21	5	5	1	1
nl.tudelft.jpacman.board		86%		58%	44	93	2	110	0	40	0	7
nl.tudelft.jpacman.sprite		86%		59%	30	70	11	113	5	38	0	5
nl.tudelft.jpacman		69%		25%	12	30	18	52	6	24	1	2
nl.tudelft.jpacman.points		60%		75%	1	11	5	21	0	9	0	2
nl.tudelft.jpacman.game		87%		60%	10	24	4	45	2	14	0	3
nl.tudelft.jpacman.npc		100%		n/a	0	4	0	8	0	4	0	1
Total	1,213 of 4,694	74%	293 of 637	54%	293	590	229	1,039	51	268	6	47

https://github.com/ZacheryEC001/cs472-team6